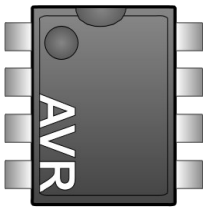


AVRPascal

Manual

version 02/28/2024

1. Introduction

AVRPascal is a Pascal code editor designed for programming AVR microcontrollers (ATtiny and ATmega families). It leverages the FreePascal[1] compiler to compile source code and AVRdude[2] to transfer the compiled code to the microcontroller's memory using programmers like USBasp[3] or Arduino Uno[4] (ATmega328p only). Key features:

- Syntax highlighting for improved code readability.
- Error indication: Points to the line of code containing compilation errors.
- Tooltips: Provide information about function and procedure parameters, record fields, and class methods.
- Tabbed document interface for efficient code organization.
- Microcontroller fuse-bit configuration capability.

AVRPascal is a 64-bit application. It is available in three versions for the following operating systems: Windows, Linux (Debian-based distributions), and MacOS. The functionality of these versions is almost identical, with minor differences arising from the specific requirements of each operating system.

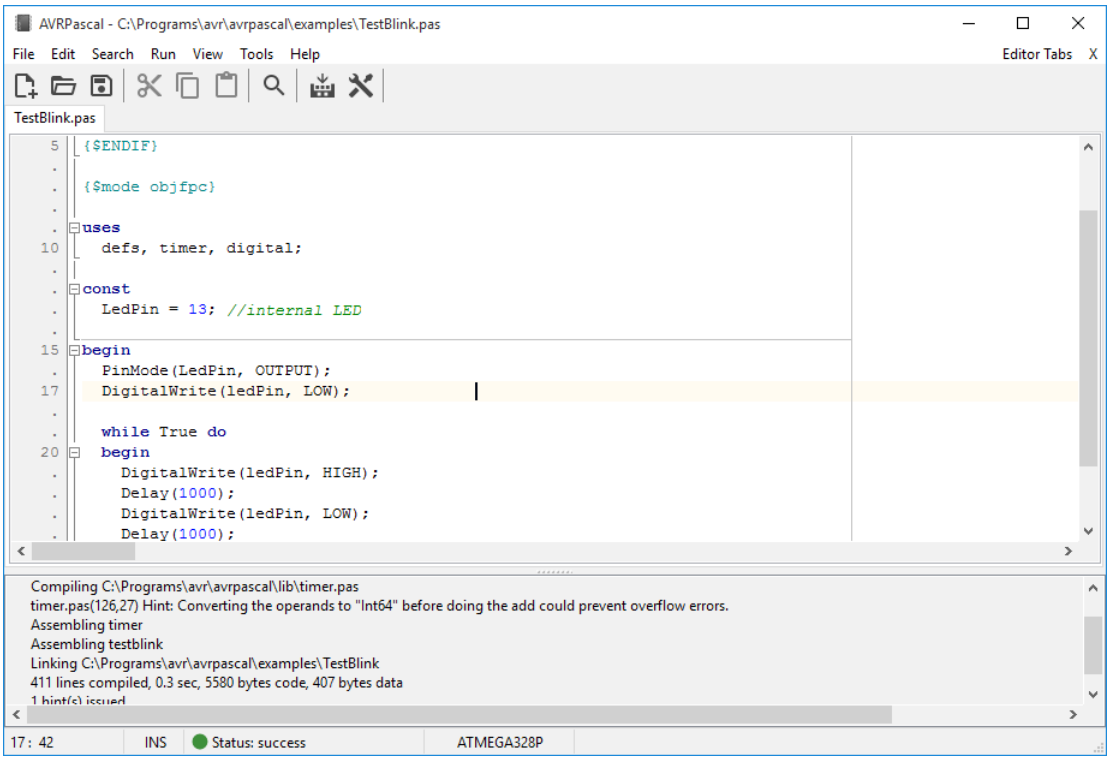
2. Installation

The application should be installed in the target operating system using using the appropriate installer. On Windows use the *.exe* file. You may need to install additional drivers for the USBasp programmer (*ibusb*[5] library driver) or Arduino Uno[6] board. On Linux (Debian-based distributions) use the *.deb* package. Additional drivers are usually not required as they're included in the operating system. For USBasp or Arduino, ensure AVRPascal has administrator privileges or add your user to the *dialout* group. On MacOS use the *.pkg* file. No additional procedures should be necessary.

1 https://www.freepascal.org/
2 https://savannah.nongnu.org/projects/avrdude/
3 https://www.fischl.de/usbasp/
4 https://store.arduino.cc/products/arduino-uno-rev3/
5 http://zadig.akeo.ie/
6 https://docs.arduino.cc/tutorials/generic/DriverInstallation/

3. Editor

The application window, similar to other code editing and compiling applications, is divided into several areas: menu bar, a toolbar, a tabs area, a messages area, and a status bar. The application allows you to edit multiple files in tabs, with at least one editor tab always open. The message list includes notifications from the compiler, uploader and debugger (if installed).

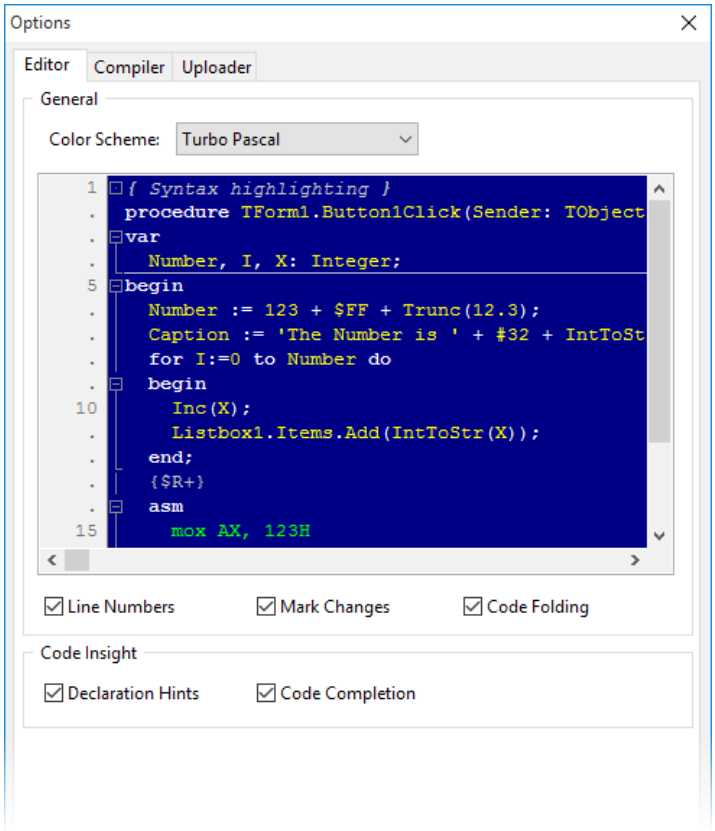


The application's menu offers standard editor functions for opening, editing, and saving Pascal source files (.pas, .pp, .inc). You can also search within the edited file. Additionally, standard text editing functions like selection, cutting, copying, pasting, deleting, and undo/redo are available.

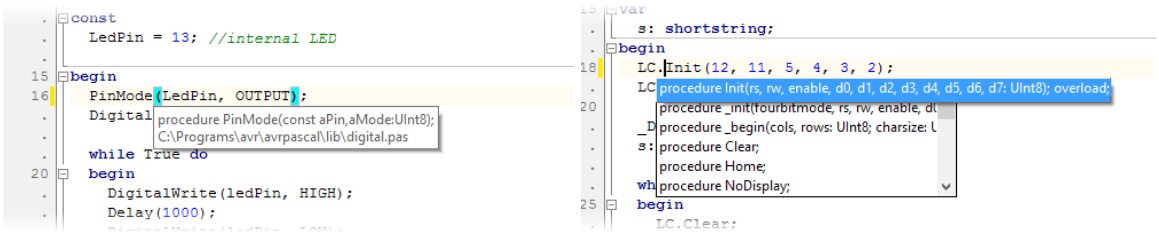
The main feature of AVRPascal that makes working with source code easier is Pascal syntax highlighting. Currently, the application supports two schemes: *Delphi* and *Turbo Pascal*, referring to the colors used in these applications. The default color scheme is the *Delphi* scheme. It can be changed by selecting the *Options* menu function and going to the *Editor* tab. Here you can also define the features of the editor's sidebar (gutter): visibility of line numbers, color highlighting of lines that have been changed and the ability to code folding. Additionally, it is possible to export the source code to an HTML file, including syntax coloring (*File->Export to HTML*)

AVRPascal simplifies working with source code through its Pascal syntax highlighting. It currently supports two color schemes, *Delphi* and *Turbo Pascal*, mimicking the styles used in those applications. The default scheme is Delphi, but you can change it through the *Options* dialog window *Editor* tab (menu *View->Options*). Here, you can also configure the gutter (area on the left side of the editor) features, including line number

visibility, highlighting of changed lines, and code folding. Additionally, you can export source code with syntax highlighting to an HTML file (*File->Export to HTML*)[7].



AVRPascal further simplifies your work with source code through two features: declaration hints and code completion (list of record type fields and class type methods). These features, common in code editors, provide suggestions as you type. Parameter hints appear after opening a parenthesis (bracket), while a list of record fields and class methods pops up after a dot.



For increased efficiency, AVRPascal provides a set of keyboard shortcuts for working with source code:

Shortcut	**Function**
Ctrl+N	Opens a new editor tab (*File->New*)

7 The MacOS version does not include the ability to export Pascal source files to HTML format.

Ctrl+O	Opens the source file (*File->Open*)
Ctrl+S	Saves the source file (*File->Save*)
Ctrl+Z	Undoes the last change in the code (*Edit->Undo*)
Ctrl+X	Cuts the selected text and places it in the clipboard (*Edit->Cut*)
Ctrl+C	Copies selected text (*Edit->Copy*)
Ctrl+V	Pastes the selected text (*Edit->Paste*)
Ctrl+A	Selects the entire text of the document (*Edit->Select All*)
Del	Deletes the selected text or character after the cursor
Ins	Switches between insert and overwrite characters
Ctrl+F	Displays a search dialog for the entered characters in the text (*Search->Find*)
F3	Finds the next occurrence of the entered characters in the text (*Search->Find Next*)
Ctrl+/	Comments on the selected text using “//” characters or removes the comment
Ctrl+U	Increases the indentation of selected text by two spaces
Ctrl+I	Decreases the indentation of selected text by two spaces
Ctrl+Up arrow	Moves the cursor to the procedure/function/method declaration
Ctrl+Down arrow	Moves the cursor to the first line of procedure/function/method code
Ctrl+[text under the mouse cursor]	When clicked, it searches the current document and other documents defined in the uses section for text declarations “under" the mouse cursor. Equivalent to the *Find declaration* context menu function.

4. Compiling your programs

AVRPascal uses the FreePascal compiler version 3.2.2. To compile the code, FreePascal translates Pascal code into assembly code and then utilizes the GNU toolchains for AVR microcontrollers, which includes the avr-as assembler and the avr-ld linker. This process generates three output files: *.bin (binary), *.elf (linker format), and *.hex (text format). Due to FreePascal's capabilities, the list of supported AVR microcontrollers is quite wide, encompassing the ATtiny and ATmega families:

ATMEGA168PA	*ATMEGA3290A*	*ATMEGA645P*	*ATTINY24*
ATMEGA169A	*ATMEGA3290P*	*ATMEGA649*	*ATTINY24A*
ATMEGA169P	*ATMEGA3290PA*	*ATMEGA6490*	*ATTINY25*
ATMEGA169PA	*ATMEGA329A*	*ATMEGA6490A*	*ATTINY26*
ATMEGA16A	*ATMEGA329P*	*ATMEGA6490P*	*ATTINY261*
ATMEGA16HVB	*ATMEGA329PA*	*ATMEGA649A*	*ATTINY261A*

ATMEGA16M1	*ATMEGA32A*	*ATMEGA649P*	*ATTINY28*
ATMEGA16U2	*ATMEGA32C1*	*ATMEGA64A*	*ATTINY4*
ATMEGA16U4	*ATMEGA32HVB*	*ATMEGA64C1*	*ATTINY40*
ATMEGA2560	*ATMEGA32M1*	*ATMEGA64M1*	*ATTINY4313*
ATMEGA2561	*ATMEGA32U2*	*ATMEGA8*	*ATTINY43U*
ATMEGA32	*ATMEGA32U4*	*ATMEGA8515*	*ATTINY44*
ATMEGA324A	*ATMEGA48*	*ATMEGA8535*	*ATTINY44A*
ATMEGA324P	*ATMEGA48A*	*ATMEGA88*	*ATTINY45*
ATMEGA324PA	*ATMEGA48P*	*ATMEGA88A*	*ATTINY461*
ATMEGA325	*ATMEGA48PA*	*ATMEGA88P*	*ATTINY461A*
ATMEGA3250	*ATMEGA64*	*ATMEGA88PA*	*ATTINY48*
ATMEGA3250A	*ATMEGA640*	*ATMEGA8A*	*ATTINY5*
ATMEGA3250P	*ATMEGA644*	*ATMEGA8U2*	*ATTINY828*
ATMEGA3250PA	*ATMEGA644A*	*ATTINY10*	*ATTINY84*
ATMEGA325A	*ATMEGA644P*	*ATTINY13*	*ATTINY84A*
ATMEGA325P	*ATMEGA644PA*	*ATTINY13A*	*ATTINY85*
ATMEGA325PA	*ATMEGA645*	*ATTINY1634*	*ATTINY861*
ATMEGA328	*ATMEGA6450*	*ATTINY167*	*ATTINY861A*
ATMEGA328P	*ATMEGA6450A*	*ATTINY20*	*ATTINY87*
ATMEGA329	*ATMEGA6450P*	*ATTINY2313*	*ATTINY88*
ATMEGA3290	*ATMEGA645A*	*ATTINY2313A*	*ATTINY9*

Compiler settings can be found in the *Compiler* tab of the *Options* window (accessible through the *View->Options* menu). These settings are global, meaning any source file compiled in AVRPascal will use the configurations defined here.

In the *Device* frame, choose the target microcontroller type. This ensures specific definitions are included during compilation for the selected microcontroller[8]. The *Rebuild RTL from source if needed* option determines whether AVRPascal automatically rebuilds Free Pascal runtime library (RTL) files after changing the microcontroller type. If unchecked, AVRPascal prompts you for confirmation.

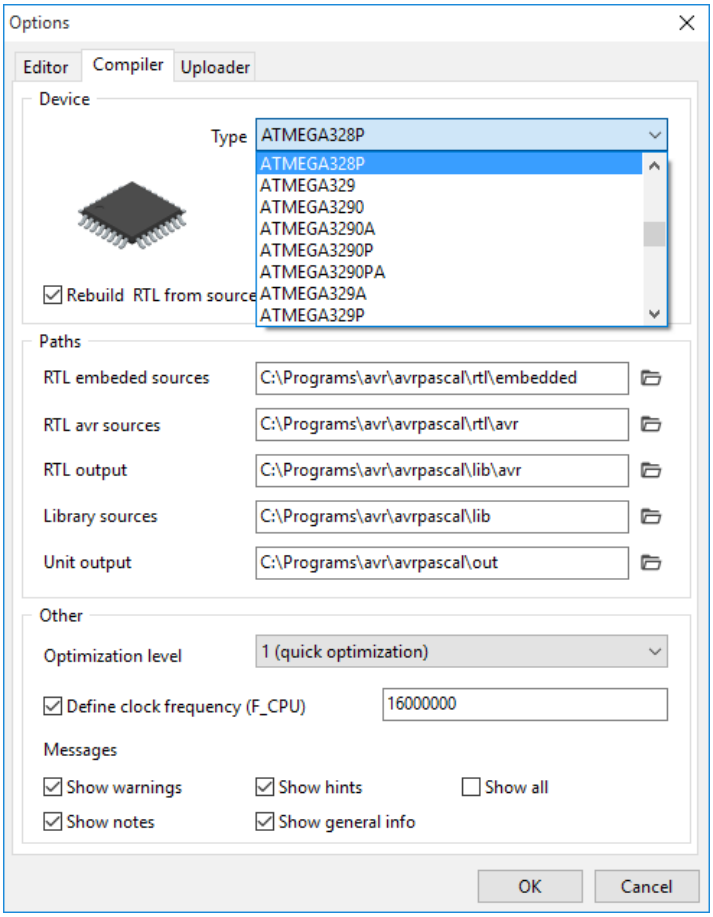
The *Paths* frame manages the locations for files used by FreePascal during compilation. Paths defined in *RTL avr sources*[9] and *RTL embeded sources*[10] determine the source files for the FreePascal runtime library (RTL), which provide essential functionalities used by compiled programs. *RTL Output* and *Library Sources* paths specifie the locations

8 The current microcontroller type is also displayed on the status bar.

9 The *RTL avr sources* path is used to store definition files for individual types of AVR microcontrollers.

10 The *RTL embedded sources* path is used to store FreePascal "system" files from which the system.o file is builded.

where the compiled versions of the RTL files or other library files[11] are saved. *Unit Output* path determines where the compiled code for your program files (units) will be placed. The paths are automatically set during AVRPascal installation and changing them is not recommended for beginners as it can affect compilation processes.



The *Other* frame includes additional compiler settings like optimization level[12], target microcontroller's clock frequency, and the verbosity level of compiler messages displayed in the AVRPascal messages area.

As mentioned earlier, AVRPascal utilizes global compiler settings and doesn't rely on project files or additional definition files. Consequently, it uses the traditional Pascal source file types: defined by *program* keyword, for creating executable files, and defined by *unit* keyword, for creating module files. Moreover during compilation AVRPascal defines a symbol corresponding to the global microcontroller type (e.g. *atmega328p*, see the list above) and a symbol indicating the family it belongs to (*familytiny* or *familymega*). These symbols enable features like conditional compilation (include or exclude code segments based on the chosen microcontroller) or defining compilation errors (generate specific error messages based on the microcontroller type), e.g.:

11 The *Library sources* path is used to point to the sources of AVRPascal's UnoLib library.

12 Using high build optimization levels is not recommended, as it may increase the chance of encountering compilation errors or issues with correctly compiling the source files.

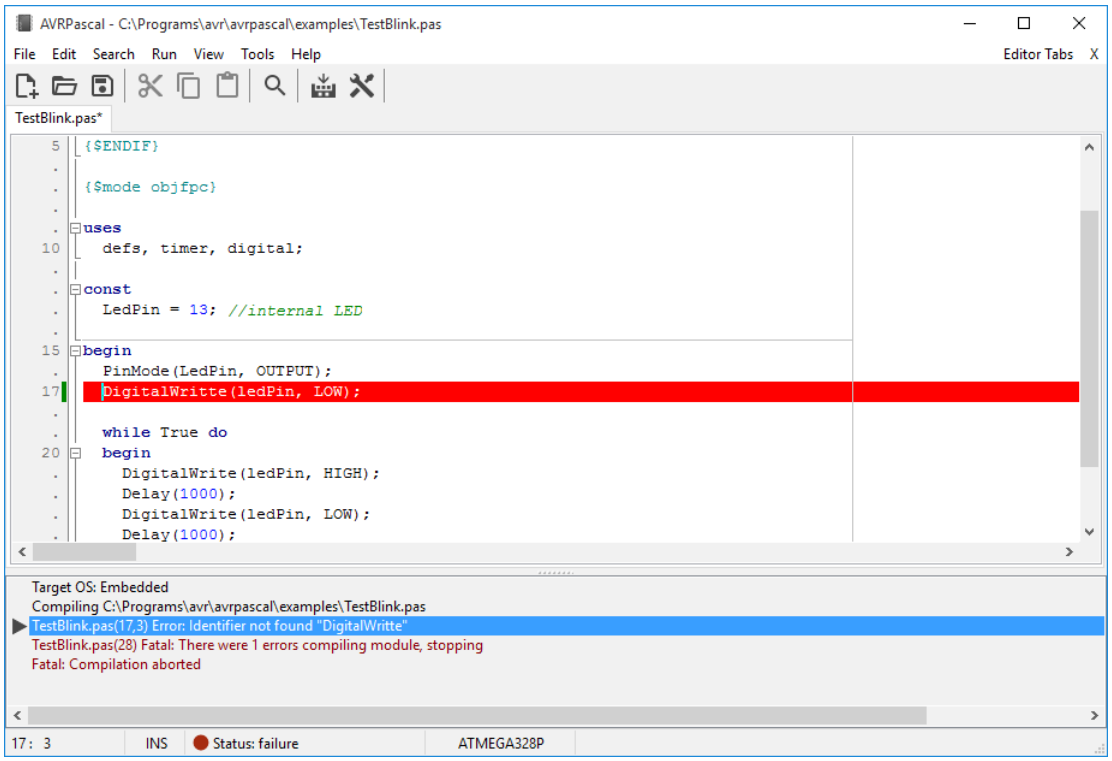
```
{$IFNDEF atmega328p}
 {$Fatal Invalid controller type, expected: atmega328p}
{$ENDIF}
```

In this example, if the global microcontroller type is other than *atmega328p*, compilation of the source file will be interrupted and an error message *Invalid controller type, expected: atmega328p* will be displayed in the messages area.

Compilation and execution commands for the currently active editor tab are located in the *Run* menu. They allow compilation (*Compile*), build (*Build All*), deletion of compilation result files (*Clear All*) and rebuilding RTL files depending on selected microcontroller (*Rebuild RTL Sources*). The *Upload* function allows you to transfer the compiled program to the target microcontroller's flash memory using the uploader. This enables running the program directly on the device. AVRPascal also offers keyboard shortcuts to streamline compilation and execution processes:

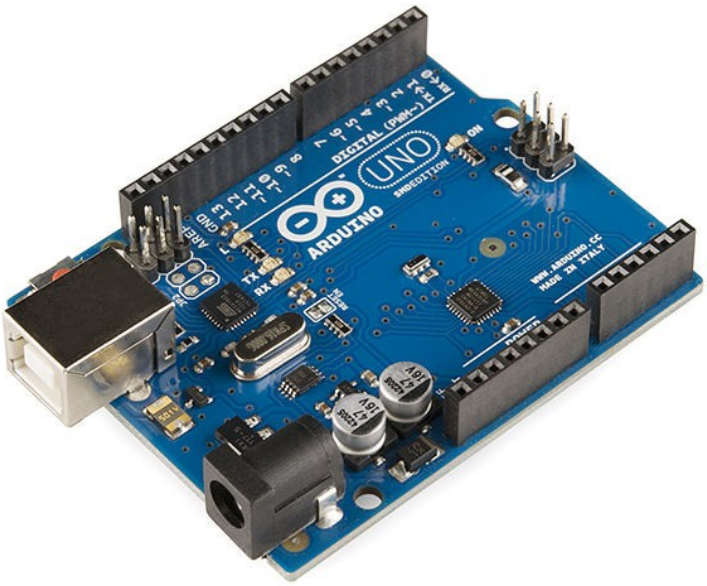
Shortcut	**Function**
Ctrl+F9	Compiles the source file from the active tab (*Run->Compile*)
Shift+F9	Builds the source file from the active tab (*Run->Build All*)
F9	Exports the compiled source file to the microcontroller's flash memory via the uploader (*Run->Upload*)

If the compilation was successful, the status bar will display 🟢*Status: success*, otherwise 🔴*Status: failure*. Additionally, if the compiler indicated an incorrect place in the source code, the line with the error will be highlighted in red.



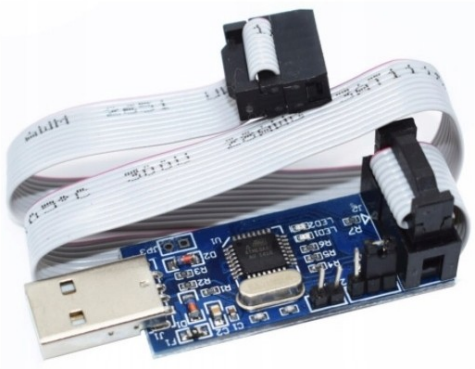
5. Using the programmer and uploader

To run the compiled source code, it must be loaded into the microcontroller's flash memory using a programmer, which is a physical device that communicates with the microcontroller. The programmer is typically connected to the computer via a USB port[13]. An uploader, which is a program that sends binary data to the programmer, is used to transfer the compiled code. AVRPascal uses AVRDude as an uploader, which supports most AVR microcontrollers and works with both the USBasp programmer and Arduino Uno.

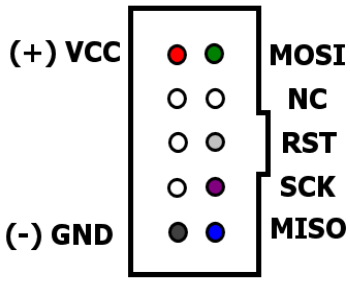


Arduino Uno

In the case of Arduino Uno used as a programmer, simply connect the device to the computer via the USB port. The limitation of this solution is the fact that Arduino Uno has a built-in ATmega328p chip, hence when Arduino is used as an AVRPascal programmer, it can only use sources intended for this microcontroller.



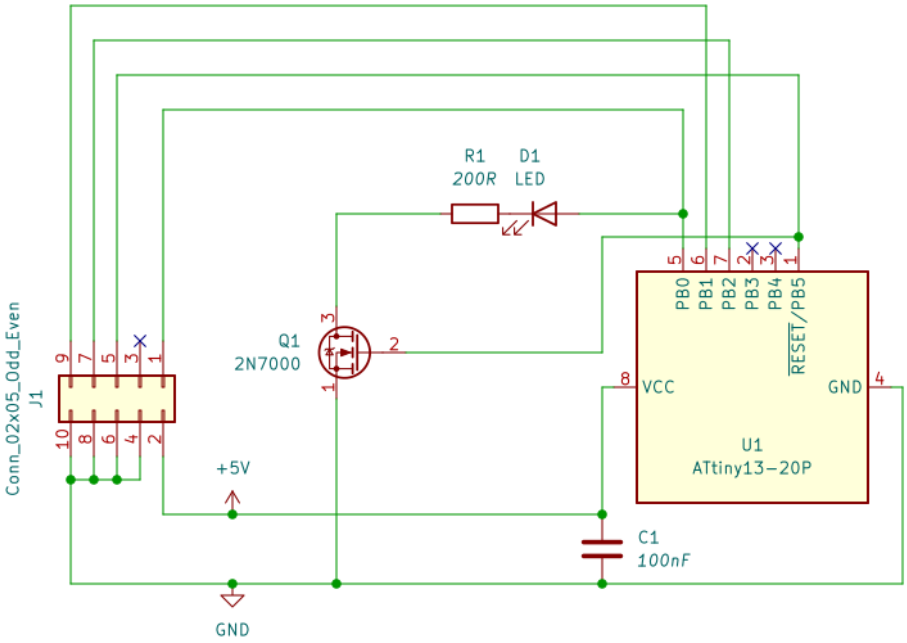
USBasp programmer with tape



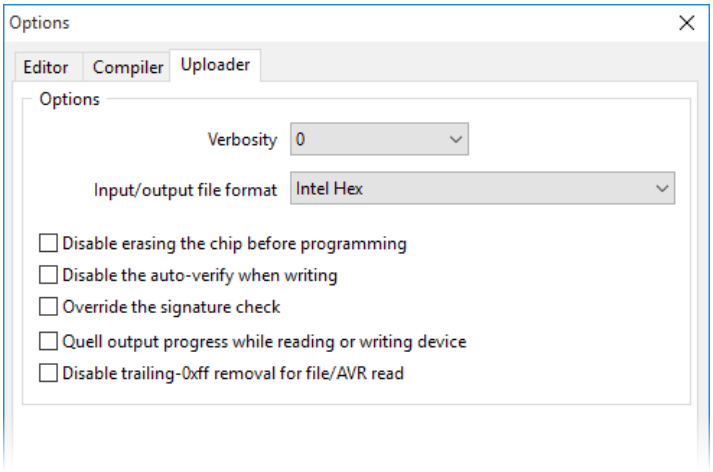
USBasp programmer connector

13 There are other methods used in older generations of PCs, using e.g. the COM or LPT port, but these solutions are impractical because the standard is currently the USB port and few computers have a COM or LPT connection.

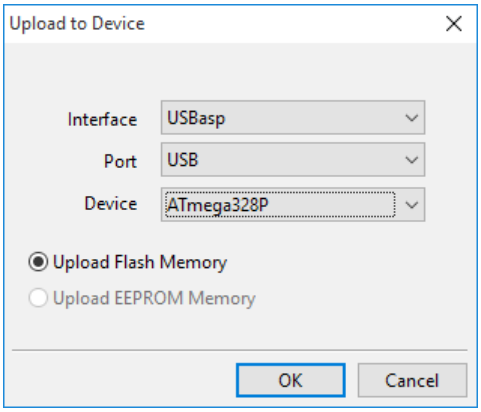
In the case of the USBasp programmer, you must ensure the correct connection of the signal lines (MISO, MOSI, RST, SCK) and power supply (VCC, GND) to the microcontroller. The method of connecting the pins of specific microcontrollers should be checked in their datasheets. For ATtiny13 it may be similar to the following:



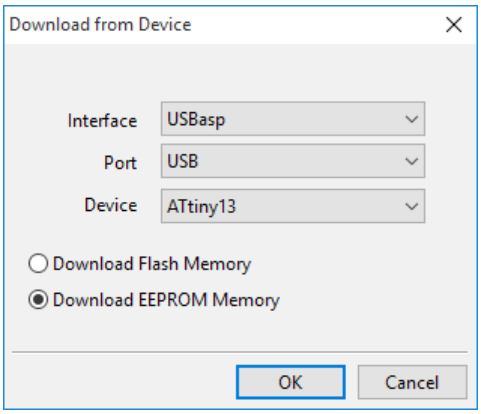
Here, the programmer should be connected to a 10-pin IDC10 (Kanda) connector, pins 4, 6, 8 and 10 of which are connected to ground, pin 2 to the 5V power supply, pin 1 to the PB0 port, pin 5 to PB5, pin 7 to PB2, pin 9 from PB1 of the microcontroller, while pin 3 remained unconnected. The circuit uses a 2N7000 MOSFET transistor that opens if the gate voltage exceeds approximately 2 V. During programming, USBasp shorts pin 1 (PB5) of the microcontroller to ground, closing the transistor and cutting off the line connected to the drain, in this case the LED. The transistor therefore plays the role of a switch that cuts off the ground line of the diode (or other elements connected in this way) during programming, and it does not negatively affect the operation of the programmer. On the other hand, pin 1 (PB5) can be used during normal operation of the microcontroller.



The uploader parameters are located in the *Uploader* tab of the *Options* window (*View->Options* menu). There you can specify the level of detail of the uploader messages displayed in the messages area (level 0 to 4), the format of the input and output files (*.bin, *.elf or *.hex) and the details of AVRdude's operation when programming or reading data from the microcontroller. These include: disabling data clearing mode on the microcontroller before programming, disabling write verification, skipping microcontroller signature checking, visibility of read/write progress from/to the microcontroller, and removing trailing 0xFF characters when reading from a file/microcontroller[14].



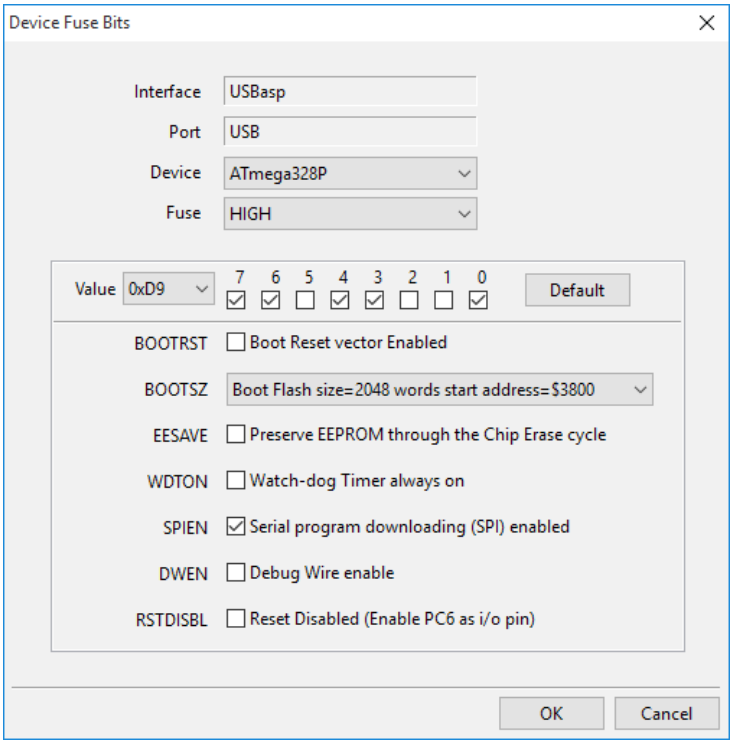
The uploader used in AVRPascal is primarily responsible for saving the compiled program's code (the effect of source compilation) to the microcontroller's flash memory. This can be done using the *Run->Upload* command (see Chapter 4). After running this command, a dialog box appears. The *Interface* and *Port* fields are only available if the current microcontroller type is set to ATmega328p. This is because the ATmega328p allows programming using both USBasp and Arduino. In other cases, the *Interface* field is set to USBasp and the *Port* field is set to USB. The *Device* field displays a list of subtypes of the selected microcontroller that are supported by the uploader. Similar to working with the compiler, uploader messages are displayed in the messages area and the result of the upload process appears in the status bar.



AVRPascal also allows you to import flash memory or EEPROM of the microcontroller to a file (menu *Tools->Download to File*) and export data from the selected

14 In most cases, when working in AVRPascal, changing the uploader parameters is not necessary.

file to flash memory or EEPROM of the microcontroller (menu *Tools->Upload from File*). These features utilize the global microcontroller type and supported file format settings you previously configured in the *Options* window.



Moreover, via the *Tools* menu it is possible to read and write the values of the microcontroller's fuses. If they are read from the microcontroller (*Tools->Get Device Fuses*), their appropriate values (low/high/extended) will appear in the messages area of the application, while their writing (*Tools->Set Device Fuses*) is made via an extended dialog box, allowing you to change individual bits. When specifying fuse-bit values other than the default ones, you should be especially careful and use the descriptions in the datasheet of individual microcontroller, because accidental settings may prevent further programming of the microcontroller using USBasp.

6. UnoLib library

UnoLib is a Pascal library designed for the Arduino Uno platform (which utilizes the ATmega328p microcontroller with a 16 MHz clock). This library is a translation of a subset of the standard Arduino libraries, adapted as needed for a Pascal environment. Moreover, support for fixed point numbers has been added. Modules included:

- *analog.pas* - support for analog pins
- *defs.pas* - definitions of constants, bit manipulations, port support
- *dht.pas* - support for DHT11/22 sensors
- *digital.pas* - support for digital pins
- *ds1302rtc.pas* - support for ds1302 real time clock
- *fix16.pas* - support for fixed point numbers

- *hardwareserial.pas* - support for serial communication
- *liquidcrystal.pas* - support for LCD
- *timer.pas* - time-related routines

7. *Freeware* version

AVRPascal is available for free download on www.akarwowski.pl. It means that it can be used completely free for home use, while it is forbidden to introduce any modifications or derive financial benefits from the distribution of the program by third parties. Although, I have made my best to ensure that the program works correctly, I cannot give any guarantees for the program operation and I am not liable for any possible damage resulting from its usage.

Andrzej Karwowski

e-mail: ackarwow@gmail.com